行列群に対するSchreier-Simsアルゴリズム

ヘンリク・バーンヒエルム

URL: http://matrixss.sourceforge.net/
Eメールアドレス: henrik.baarnhielm@imperial.ac.uk

概要本書は、行列群に特化したSchreier-SimsアルゴリズムをGAPで新たに実装することを目的としたプロジェクトの報告である。標準的なSchreier-Simsアルゴリズムについて詳細に説明し、続いて確率的Schreier-SimsアルゴリズムとSchreier-Todd-Coxeter-Simsアルゴリズムについて説明する。そして、我々の実装といくつかの最適化について議論し、最後に、GAPにおける既存の実装と比較した我々の実装の性能について報告し、ベンチマーク結果を示す。結論として、我々の実装の方が高速であり、メモリ消費量も少ない場合がある。

内容

序文

本報告書は、英国ロンドンのインペリアル・カレッジ・オブ・サイエンス・テクノロジー・アンド・メディスン（Imperial College of Science, Technology and Medicine）数学科の学生プロジェクトの一環として作成されたものである。本報告書は、純粋数学における理学修士の学位取得要件の一部を満たすものである。

指導教官であるインペリアル・カレッジのアレクサンダー・イワノフ教授とロンドン大学クイーン・メアリー校のレナード・ソイチャー博士のご指導とご協力に感謝いたします。

また、このプロジェクトを私に提案してくれた米国フォートコリンズにあるコロラド州立大学のアレクサンダー・ハルプケには、特に感謝している。

第1章

はじめに

もう1つの部分は、コンピュータ・システムGAP（[GAP]を参照）用のソフトウェア・パッケージである。この文章では、使用されたアルゴリズムやその複雑さなど、パッケージの基礎となる数学について述べ、また、どのようなデータ構造が使われたか、どのように実装されたかなど、よりコンピュータサイエンス的な側面についても述べる。

計算群理論(CGT)は群論と計算機科学の境界に位置する研究分野であり、CGTの研究はしばしば理論的(数学的)と実践的(プログラミング)の両方の性質を持ち、理論的な結果(数学的定理と証明)と実践的な結果(ソフトウェア)の両方をもたらすが、このプロジェクトも例外ではない。CGTの入門的なサーベイは、[Sim03]、[Ser97]、[Neu95]、[CH92]にある。

このプロジェクトの目的は、行列群に対するSchreier-Simsアルゴリズムの実装を持つGAPパッケージを作ることである。Schreier-Simsアルゴリズムは群の基底と強生成集合を計算するもので、この基本的なアルゴリズムの実装はすでに標準的なGAPディストリビューションに含まれている。このプロジェクトのアイデアは、行列群に注目し、行列を直接扱うアルゴリズムのバージョンを実装し、この方法でより効率的な実装が得られるかどうかを確認することである。

計算行列群論のサーベイが[NP01]にある。我々は有限群、すなわち有限体上の行列群にしか興味がないので、計算不可能性や決定不可能性の問題を心配する必要はないことに注意すべきである。

まず、Schreier- Simsアルゴリズムについて説明する前に、使用されている群論と計算機科学の基本概念について簡単に触れる。この説明は非常に詳細であり、その後、このプロジェクトで実装されているアルゴリズムの変種、ランダム（つまり確率的）Schreier-SimsアルゴリズムとSchreier-Todd-Coxeter- Simsアルゴリズムについて説明する。その後、実装について述べ、アルゴリズムを高速化するために行われたいくつかのトリックや改良について述べる。最後に、この実装の実用的な性能とベンチマーク結果

を示し、GAPにおける既存の実装と比較する。

これに似た報告として[Mur93]がある。
かなりの量のインスピレーションが湧いてくる。

第2章

予備知識

このセクションの定義と記述は既知であることを前提としているが、著者はしばしば異なる表記法を用い、以下の概念のいくつかに若干異なる意味を持たせることがあるため（例えば、グラフの厳密な定義は異なる傾向がある）、ここではとりあえずそれらを記す。

1. グラフ理論

Big89]に従う。

定義2.1. *有向グラフは* $G = (V, E)$の順序ペアであり、VはG*の頂点*、E⊆V×VはG*の辺を表す*有限の非空集合である。

備考2.2.2.1の意味でのグラフは，[BH99]の意味での*計量グラフ*ではないことを強調するために， 文献では*組合せグラフと*呼ばれることがある．我々は有限グラフにのみ興味があり、幾何学には興味がないので、この命名法は使わない。

備考2.3.この定義から、我々のグラフは多辺を持たないが、ループを持つ可能性がある。我々が興味を持つグラフはこれだけなので、以後「有向」という言葉を省略する。

定義2.4.$_{12}G = (V, E)$がグラフであるとき、i = 1, ...v_i , v_{i+1}) ∈ E for i = 1, ... , n - 1 *と*なるようなGの頂点のシーケンスv = v_n = u。 n - 1は、vからEへの *ウォークと*呼ばれる。
u.歩幅$v = v_1 , . . .v_k = u$*の長さは*k - 1である。$v = u$ ならば，その歩みは*サイクルと*呼ばれる．歩行のすべての頂点が異なる場合，それは*パスと*呼ばれる．G*の*すべての頂点の対がパスで結ばれるならば、Gは*連結している*。

定義2.5.グラフ$G' = (V' , E')$ は、以下の場合、グラフ$G = (V, E)$の*部分グラフ*である。

$V' \subseteq V$、$E' \subseteq E$。

定義2.6.木はサイクルのない連結グラフである。*根付き*木とは、頂点$r \in V$がルートとして指定された木$T=(V, E)$のことである。

定義2.7.$G=(V,E)$がグラフであるとき、木Tは以下の*スパニングツリー*である。TがGの部分グラフであり、Tが頂点集合Vを持つ場合。

以下は初歩的なものであり、証明は省略する。

命題2.8.*木$T=(V, E)$において、*｜E｜=｜V｜-1｜*であり、すべての異なる頂点の対の間に一意なパスが存在する。逆に、Gが、異なる頂点の組がすべて一意なパスで結ばれるグラフであるならば、Gは木である。*

命題2.9.*すべてのグラフはスパニングツリーを持つ。*

定義2.10.$T = (V, E)$ を根 $r \in V$ を持つ根付き木とする。ノード$x \in V$の深さは、xからrへのパスの長さである。

定義2.11.$G = (V, E)$がグラフであるとき、Gのラベリングとは関数w：$E \to L$ ここで、L はある「ラベル」の集合である。ラベル付きグラフとは、 対応するラベルを持つグラフである。

2. グループ理論

ここで、標準的な参考文献[BB96]と[Ros94]に従った群論をいくつか紹介しよう。

注2.12.本レポートではすべてのグループが有限であると仮定している。

定義2.13.$G = \langle$S↪Pe_27E9 (S)が群であるとき、ケイリーグラフ$C_G (S)$は頂点集合Gと辺$E = \{(g, sg) \mid g \in G, s \in S\}$ を持つグラフである。

定義2.14.有限集合X上の群Gの作用とは、$\lambda : G \to \mathrm{Sym}(X)$ （$\mathrm{Sym}(X)$はX上の順列群）の同型性である。λが帰納的であれば、作用は忠実である。

備考2.15.計算群論の慣例に従い、作用は右からであり、$\lambda(g)x$ は $g \in G$ および $x \in X$ に対して x^g と略される。作用がある以上、指数の規則のいくつかは成り立つことに注意。例えば、 $(x\,)^{gh} = x$ 。^{gh}

定義2.16.各点$\alpha \in X$に対して、αの軌道は$\alpha^G = \{\beta \in X \mid \beta = \alpha^g$ 、$g \in G\}$であり、αのスタビライザーは$G_\alpha = \{g \in G \mid \alpha^g = \alpha\}$である。

以下は初歩的なものであり、証明は省略する。

命題2.17Gを有限集合Xに作用する群とする。
$G_p \leq G$ となるので、帰納的に次のように定義できる。

(2.1) $$G_{\alpha 1,\alpha 2,\cdots,\alpha n} = (G_{\alpha 1,\alpha 2,\cdots,\alpha n-1})_{\alpha n}$$

ここで、$n > 1$および$\alpha_1, .\alpha_n \in X.$

命題2.18.Gを有限集合Xに作用する群とする。各$p \in X$に対して、写像$\mu_p : G/G_p \to p^G$ は次式で与えられる。

(2.2) $$G_p\, g \mapsto p^g$$

for each $g \in G$, is a bijection. In particular, $p^G = [G : G_p].$

3. コンピューター・サイエンス

計算機科学に関しては、我々の標準的な参考文献は[CLR90]である。まず、アルゴリズムの複雑さ解析とその漸近表記法、特にO(-)-表記法は

既知のものとする。また、幅優先探索、連結成分の計算、スパニングツリーアルゴリズムのような基本的なグラフアルゴリズムも知っているものとする。

ハッシュ・テーブルも同様に、これ以上説明することなく使用する。このプロジェクトのコードではハッシュを明示的に使用せず、GAPに依存しているが、筆者の知る限り、人類が知る限り最高の汎用ハッシュ関数は[Jen97]に記述されていることを述べておく。

第3章

Schreier-Simsアルゴリズム

ここで、Schreier-Sims アルゴリズムについて説明する。参考文献は[CSS99]、[Ser03]、[But91]である。最初に、いくつかの背景を説明し、このアルゴリズムが解決する問題を明確にする必要がある。

1. 背景と動機

群論における研究の全体的な目標は、群を理解し、群に関する様々な疑問に答えることである。特にCGTでは、アルゴリズム的な性質の問題に関心が集中しており、群Gが与えられた場合、以下のようなアルゴリズムに興味がある：

- G|とは?
- Gの全要素を、繰り返さずに列挙せよ。
- G≦Hで、任意のg∈H*が与えられた*とき、g∈G *であることは*正しいか？ これは*メンバーシップ問題*と呼ばれる。
- ランダムなg（G）を生成する。
- G*は* abelian（可溶性、多環性、nilpotent）か?
- Gの共役クラスの代表を見つける。
- Gの合成級数（合成因子の（同型の）級数を含む）を求める。
- g∈GまたはH≤Gが与えられたとき、それぞれgの中心化子またはHの正規化子を求めよ。

これらのタスクを達成するためには、Gのコンピュータ表現、つまりデータ構造が必要である。群を与える一般的な方法は生成集合を使うことであるが、これだけでは問題の解決には役立たないので、よりよい表現が必要である。生成集合からこの表現を計算することが可能でなければならないし、Gの部分群が何らかの直接的な方法で表現を継承すれば、アルゴリズムを設計する際に分割統治技術が使えるようになる（[CLR90]参照）。

2. ベースと強力な発電セット

Gの部分群の連鎖がある場合を考えてみよう。

(2.1) $$g = g^0 \geq g^1 \geq \cdots \geq g^n = 1$$

各 $g \in G$ は $g = g\, u_{11}$ と書くことができる。ここで u_1 は G^1 の代表 g であり、$g_1 \in G^1$ である。帰納的に g を因数分解すると $g = u\, u_{nn-1} \cdots u_1$ となる。亜群連鎖は1に達するからである。さらに、この因数分解は、*各iについて*G^{i+1} のコセットがG^i を分割するので*一意*であり、同じ理由で、異なる群要素は異なる因数分解を持つ。

We thus see that if we know generating sets for the subgroups in such a subgroup chain, and if we know a right transversal T^i of the cosets of G^{i+1} in G^i for each $i = 0, \ldots, n-1$, then we could easily solve at least the first two listed problems listed in section 1. By Lagrange we know $|G| = T^1\, G^1$ and inductively $|G| = T^1 \cdots |T^n|$, which solves the first problem. Using the factorization we have a bijection from G to $T^1 \times T^2 \times \cdots \times T^n$ so by enumerating elements of the latter set and multiplying in G we can list the elements of G without repetition.

ここで、特殊なタイプの部分群連鎖を紹介する。

定義3.1.$G_{\alpha\, 1,\ldots,\alpha n} = 1$となるような$X$の点の列$(\alpha_1, \ldots, \alpha_n)$を$G$の*基底*と呼ぶ。

(2.2) $$G \geq G_{\alpha} 1 \geq \cdots \geq G_{\alpha\, 1,\ldots,\alpha n} = 1$$

$G_i = G_{\alpha,\ldots,\alpha}$とす　$i = 1, .n.$ $S \cap G^i =$ となるようなGの生成集合S。
る。
i
すべてのiについてG^i をGの*強生成集合*（SGS）と呼ぶ。

基底と強生成の概念は[Sim70]で順列群の文脈で初めて導入されたもので、主に順列群に対してではあるが、基本的に重要である。我々の問題の最初の2つは、部分群連鎖が分かれば解けることが既に分かっており、基底と強生成集合を使えば、メンバーシップ問題も簡単に解けることが分かるだろう。Schreier-Simsアルゴリズムは，生成集合Sが与えられた群Gの基底集合と強生成集合を計算するもので，Gが順列群であれば効率的なアルゴリズムであるため，基底集合と強生成集合の概念は非常に重要になっている．順列群に対するより洗練されたアルゴリズムの多くは、入力として基底集合と強生成集合を必要とする。一方、行列群については、後で見るように、状況はもう少し複雑である。

3. シュライヤーの木

Schreier-Simsアルゴリズムそのものを説明する前に、いくつか説明すべき補助的なアルゴリズムがある。したがって、有限集合Xに作用する群$G = \langle S \langle$を考えよう。前節から、Gの基底（$\alpha_1, \ldots, \alpha_n$）があり、対応する安定化鎖$G \geq G^1 \geq \cdots G^n = 1$があったとしても、$i = 1, \ldots, n-1$の$G^i$ において、G^{i+1} のコセットの右横線を求める必要があることがわかる。, n - 1.しかし、これらの群はX上の作用からスタビライザーであるので、命題2.18とし、代わりに軌道$\alpha, \alpha_1^G\; {}_2^{G^1}, \ldots, \alpha_n^{G^{n-1}}$.この軌道は計算は簡単だ。

X上のGの作用は、定義2.13のGのケイリーグラフ$C_S(G)$に類似した、S

*からの*ラベルを持つラベル付きグラフで表すことができる。グラフの頂点をX、辺を{(p, p^g) | p∈X, g∈G} とする。辺(p, p^g)はg でラベル付けされる。作用の軌道がこのグラフの連結成分であることは明らかで、α^G はα_1 を含む成分である。我々はこの軌道を見つけ、それをデータ構造に格納することに興味がある。したがって、連結成分のスパニングツリーを考えることになる。残されたグラフの辺は、G*の*作用に関する追加的な関連情報を与えることはなく、 点間を移動する代替的な方法を与えるだけであり、木はグラフよりも保存がかなり簡単である。

定義3.2.Gを有限集合Xに作用する群とし、$\alpha \in X$とする。αを含む対応するグラフの成分に対してαを根とするスパニングツリーを、軌道α^Gに対する*シュライヤー木*と呼ぶ。

シュライヤー木は、αを含む成分の単純な幅優先探索によって計算することができ、こうして軌道を求めるアルゴリズムができる。しかし，グラフを明示的に生成し，それから連結成分を計算し，最後にシュライヤー木を求めるというのは，もちろん計算上良いアイデアではない．アルゴリズム3.1が示すように、グラフそのものがなくても、幅優先探索でシュライア木を求めることができる。

アルゴリズム 3.1：ComputeSchreierTree

データデータ：有限集合Xに作用する群$G = \langle S \rangle$と点$\alpha \in X$。
結果：αのシュライアツリー^G。
/*また、関数AddChild(T, p_1, p_2, l)は、ラベルlを持つ木Tのp_1に子としてp_2を追加する。 */

```
開始
    ポイント := {α}
    tree := Tree(α)
    繰り返す
        children :=∅
        各p∈ポイントは次のようにする。
            各s∈S do
                p′ := p^s
                if p′ 木
                    AddChild(tree, p, p′, s)
                    children := children∪ {p′}.
                終わり
            終了
        終了
        ポイント := 子供たち
    ポイント＝φまで
    リターン ツリー
エンド
```

前述したように、我々はコセット代表の代わりに軌道を保存するために命題2.18を使ったが、後者はまだ必要である。幸いなことに、シ

ュライアー木があればそれは簡単にできる。ある点$\alpha \in X$の軌道α^G のシュライエ木$T = (V, E)$があるとする。もし$g \in G$ならば、$\alpha^g \in V$ であり、Tにはαからα^g への���スがある。$s_1 , s_2 , \ldots ,$ s をラベルとする。$s_i \in S$ for $i = 1, \ldots , s_n$ をパスのラベルとする。$h = s\ s_{12}$ - - s_n とすると、明らかに $\alpha^h = \alpha^g$ なので、G_α g = G_α h となり、hはgのコセット代表である。したがって、gのコセット代表を見つけるには、α^g からルートαへの一意なパスをたどり、エッジ・ラベルを乗算するだけでよい。アルゴリズム3.2は、与えられた点からルートへのパスをたどるという、少し一般的なタスクを実行する。

アルゴリズム 3.2: `OrbitElement`

データデータ：有限集合Xに作用する群$G = \langle S \rangle$、点$\alpha \in X$の軌道α^G に対するシュライア木T、および任意の点$p \in X$。
結果：$\alpha^g = p$ を満たすような要素 $g \in G$
/*T 内の p とその親との間のユニークな辺のラベルを返す関数 `EdgeLabel`(T, p) が存在することを仮定する。 */

```
開始
    g := 1
    while p /= α do
        s := EdgeLabel(T, p)
        p := p^(s^-1)
        g := sg
    終わり
    g
エンド
```

これらのアルゴリズムの複雑さの解析は後ほど行う。

4. 問題の定式化

Schreier-Simsアルゴリズムによって解決される問題をより正式に述べるには、以下が必要である。

定義3.3.Gを有限集合Xに作用する群とする。Xの点$B = (\alpha_1, \ldots, \alpha_n)$の列と、$S$のどの要素も$B$のすべての点を固定しないような$G$の生成集合$S$を、それぞれ*部分基底*および*部分強生成集合*と呼ぶ。

備考3.4.定義3.2のような基底と強い生成集合を
完全だ。

備考3.5.について、$G^i = G_{\alpha,\ldots,\alpha}$、$\mathbb{S}^i = \mathbb{S} \cap G^i$ および $H^i = S^i$ と定義する。$i = 1, \ldots n$, Sのどの要素もBのすべての点を固定しないので、$H^n = 1$であることがわかる（そして、$\langle \varnothing \rangle = 1$という慣例を使う）。したがって、次のようになる。

$$g \geq g^1 \geq \text{- - -} \geq g^n \tag{4.1}$$

$$g \geq h^1 \geq \text{- - -} \geq h^n = 1 \tag{4.2}$$

定義3.2、SとBは完全である。もしh（H）$^{i+1}$ ＝ Si+1 ＝ S∩Gi+1 では

ここで

さらに、$G^i \geqq S^i_{\alpha j^s} = H^i$ for $i = 1, . .$ nとし、nが等質であれば、次のようになる。

である。kなので、

$h = s_1 \cdots s_k$ i+1 ＝αi+1 α^h i+1 ＝αi+1 したがって

h ($H^i_{\alpha i+1}$.したがって、 αi+1 for $i = 0, ., n - 1$.

$H^{i+1} \leq H^i$

有限集合*X*に作用する群*G*と、*X*からの点を持つ部分基底Bと、部分強生成集合*S*が与えられたとき、Bが（完全な）基底であり、Sが（完全な）強生成集合であることを検証するか、Bと*S*を拡張してそれらが完全となるようにする。これがSchreier-Simsアルゴリズムで解かれる問題である。

アルゴリズムを設計する際には、[Leo80]の以下の結果を使用する。

THEOREM 3.6. *Let G be a group acting on the finite set X, and let $B = (\alpha_1, \ldots, \alpha_n)$ be a partial base and S a partial strong generating set for G. Let also $G^i = G_{\alpha_1,\ldots,\alpha_i}$, $S^i = S \cap G^i$, $H^i = \langle S^i \rangle$ for $i = 1, \ldots, n$ and $G = G^0 = H^0$. Then* 以下の記述は等価である：

(1) BとSが完成した。
(2) $G^i = H^i$ *for* $i = 0, \ldots, n$.
(3) $H^i_{\alpha_{i+1}} = H^{i+1}$ *for* $i = 0, \ldots, n-1$.
(4) $[H^i : H^{i+1}] = |\alpha_{i+1}^{H^i}|$ *for* $i = 0, \ldots, n-1$.

証明。備考3.5より、(1)と(2)は等価であることがわかる。仮に(2) 以下のようになる。

(4.3) $$H^i_{\alpha_{i+1}} = G^i_{\alpha_{i+1}} = G^{i+1} = H^{i+1}$$

i＝0、 .$n-1$であり、これはまさに(3)である。代わりに(3)を仮定し、さらに帰納的に$G^i = H^i$ (基本ケース$G = H^0 = G^0$ はOK)を仮定すると、次のようになる。

(4.4) $$G^{i+1} = G^i_{\alpha_{i+1}} = H^i_{\alpha_{i+1}} = H^{i+1}$$

したがって帰納法により、$i = 0, \ldots$ これは(2)である。

ここで(3)を仮定し、2.18から$[H^i : H^i_{\alpha_{i+1}}] = |\alpha_{i+1}^{H^i}|$、だから$H^i_{\alpha_{i+1}} = H^{i+1}$ (4)が得られる。最後に、(4)を仮定する。リマーク3.5より$H^i_{\alpha_{i+1}} \geq H^{i+1}$が得られる。したがって、再び2.18を用いれば、$|\alpha_{i+1}^{H^i}| = [H^i : H^{i+1}] \geq [H^i : H^i_{\alpha_{i+1}}] = |\alpha_{i+1}^{H^i}|$ したがって、$H^i_{\alpha_{i+1}} = H^{i+1}$。

先に観察したように、我々はしばしば生成集合の形でグループを与えられるが、Schreier-Simsアルゴリズムは、入力として部分的な基底と部分的な強い生成集合を必要とする。しかし、アルゴリズム3.3 を使えば、それらは簡単に計算できる。また、このアルゴリズムは、部分強生成集合がin-versesのもとで閉じており、恒等式を含まないことを確認する。アルゴリズム3.3で使われる関数NewBasePointが、Gが行列群である場合にどのように実装できるかを見てみよう。

5.シュライヤーの定理

Schreier-SimsアルゴリズムのSchreierという名前は、以下の結果に由来する。これは [Sch27]で初めて登場し，我々の証明は[Hal59]に由来する.

定理3.7（シュライヤーの定理）。*$G = \langle S \rangle$を群とし、HをGにおけるHのコセットの右横線とする。$g \in G$に対して、$Hg = H\overline{g}$となるような唯一の*

要素を$g^{-}\in T$とする。 すると、Hは次式で生成される。

(5.1) $$S_H = ts(\overline{ts})^{-1} \mid t \in T,\ s \in S$$

証明。一般性を失うことなく、$1 \in T$(H自身を代表するコセット)と仮定することができる。定義により、Hts = Htsは、すべての$t \in T,\ s \in S$に対して$ts(ts)^{-1} \in H$を意味する。したがって、$S_H \subseteq H$と$\langle S_H \rangle \leq H$なので、文の内容はもう一方の包含にある。

$h \in H \leqq G$とし、$\langle$S$\rangle = G$であるから、$h = s\, s_{12}$ - - s_kが成り立つ。 $s_i \in S \cup S^{-1}$ for $i = 1, . . k$. k＋1個の要素���シーケンスt_1 , t_2 , ... , t を定義する。t_{k+1} の$k + 1$個の要素を定義する。

アルゴリズム 3.3: GetPartialBSGS

データ有限集合Xに作用する群$G = \langle S \rangle$と点Bの列

Xの（空の可能性もある）。

結果：Gの部分基底B'、部分強生成集合S'。

/*p^g /= pとなる点$p \in X$を返す関数NewBasePoint(g)が存在すると仮定する。 */

```
開始
    ベース := B
    sgs := φ
    各 s ∈ S  {1}  do
        if base^s = base then
            point := NewBasePoint(s)
            base := base∪ {point}
        終了                }
        sgs := sgs ∪ s, s^−1
    終わり
    リターン（ベース、SGS）
エンド
```

$t_1 = 1$ であり、帰納的に $t_{i+1} = \overline{t s_{ii}}$ である。さらに、$a_i = t s t_{ii}^{-1}$ $_{i+1}$ にとって $i = 1, \ldots n$ とし

(5.2) $h = (t s t_{11}^{-1})(t s t_{22}^{-1})$ - - - $(t s t)_{nn}^{-1}$ $t_{n+1} = a\, a_{12}$ - - - $a\, t_{nn+1}$

2 3 n+1

ここで、$i = 1, \ldots$ について、$a_i \in \langle S_H \rangle$ であることを示す。$n, t_{n+1} = 1$であり、これは$H \leq \langle S_H \rangle$を意味する。

各 $i = 1, .n, s_i \in S$ または $s^{-1} \in S$. 最初の場合、直ちに次のようになる。を得る$_i = t s_{ii}$ $(t s)_{ii}^{-1} \in S_H$、2番目のケースでは、$Ht\, s_{i+1}^{-1} = Ht\, s\, s_{ii}^{-1}$ = を得る。

$Ht\, s\, s_{ii}^{-1}$ = Ht_i となり、$t_i = t\, s_{i+1}^{-1}$ となる。したがって $\overset{i}{=}$ ti+1s-1t-1 $\overset{i}{=}$

ti+1 s_i^{-1} t $_{i+1} s^{-1}$ $^{-1} \in S$ したがって $\in S_H \rangle$。

i H て$_i$

最後に、$h \in H$と$\langle S_H \rangle \leq H$から、$t_{n+1} = (a\, a_{12}$ - - $a)_n^{-1}$ h∈Hとなるので、次のようになる。

t_{n+1} はHのコセット代表であり、したがって$t_{n+1} = 1$である。したがって、$H \leq \langle S_H \rangle$となる。

我々の立場は、有限集合Xに作用する群$G = \langle S \rangle$があり、安定体G_α（ここで$\alpha \in X$）の生成子（通常*シュライヤー生成*子と呼ばれる）を見つけるためにシュライヤーのレンマを使いたいということである。アルゴリズム3.1を用いて軌道α^Gのシュライヤー木を計算すれば、アルゴリズム3.2を用いてG_αのコセットの横線を求めることができることがわかる。

$p \in X$に対して、$t(p) \in$ Gをアルゴリズム3.2の結果とする。定理3.7の表記法を用いると、$g^{-} = t(\alpha^g)$となり、横軸は$t(p) \mid p \in \alpha^G$となるので、$s \in S$、$p \in \alpha^G$に対して、シュライヤー生成は次式で表される。

(5.3) $$t(p)st(\alpha\ )^{t(p)s-1} = t(p)st((\alpha\ )\ )^{t(p)s-1} = t(p)st(p$$

$)^{s-1}$となり、Gの生成集合$_\alpha$は以下のようになる。

(5.4) $$t(p)st(p\ )^{s-1} \mid p \in \alpha^G, \text{ s} \in \text{S}\}.$$

5.1. 基底とSGSの計算我々の問題に戻ると、Gの部分基底$B = (\alpha_1 , , \alpha_n$ $)$ があり、それに対応する部分強生成
の集合Sを作れば、SchreierのLemmaを使って問題を解くことができる。定理3.6の表記法を用いて、(5.4)を用いて各G^i のシュライヤー生成子を計算し、それらをSに加え、BとSがまだ部分的であることを保証するために、いくつかのシュライヤー生成子が基底全体を固定する場合、Bに点を加えることもある。これが終わると、$H^i = G^i$ for $i = 1, . .$ したがって、BとSは完全である。アルゴリズム3.4を使ってシュライヤー生成器を計算することができる。

アルゴリズム 3.4: `GetSchreierGenerator`

データデータ: 有限集合Xに作用する群$G=\langle S\rangle$、点$\alpha\in X$の軌道α^G
のシュライア木T、$p\in X$、生成子$s\in S$。
結果pとsに対応するシュライヤー・ジェネレーター

開始
$t_1 :=$ `OrbitElement`(T, p)
$t_2 :=$ `OrbitElement`$(T, p)^s$
戻る $t_1 st_2$
エンド

しかし、この単純なアプローチには問題がある。(5.4)で定義される生成集合は非常に大きくなり、多くの冗長なジェネレーターを含む可能性がある。通常、安定器を生成するには、シュライアーの生成子の一部で十分である。Hal59]では、$[G : G_\alpha$] - G_α のシュライヤー世代の1つが恒等式に等しいことが示されている。例えば、ある点$p\in\alpha^G$ について、(p, p^s) がα^G のシュライアー木の辺であるとすると、シュライアー生成子 $t(p)st(p$ $)^{s-1}$ は恒等式である。したがって、自明でないシュライヤー生成子の数は、(|S| - 1)$[G : G_\alpha] + 1$ と同じくらいになる可能性があるが、そのうちのいくつかは互いに等しいかもしれない。

我々の場合、Sの代わりにS^{i-1} を使って、各G^i のシュライヤー生成子を計算する。S^{i-1} は、G^{i-1} の計算されたSc のシュライヤー生成子そのものであるため、G^i の非自明なシュライヤー生成子の数は、以下のように大きくなる可能性がある。

(5.5) $$1 + (|S| - 1) \prod_{j=0}^{i} |\alpha G_{j+1}|$$

軌道の大きさは｜X｜によってのみ限定されるので、(5.5)は｜X｜において指数関数的であることがわかる。
|したがって、この方法は効率的ではないかもしれない。

5.2. Reducing the number of generators. It is possible to reduce the number of generators at each step, so that our generating set never grows too large. This can be done using Algorithm 3.5, which is due to Sims, and which can also be found in [Sim03] and [CSS99].

このアルゴリズムは、生成集合をサイズ$|X|$ ∈O($|X|^2$)に縮小する。このアルゴリズムを使えば、アルゴリズム3.6とアルゴリズム3.7を使って問題を解くことができる。しかし、これはSchreier-Simsアルゴリズムではなく、同じことを実行する、より巧妙で効率的な方法である。

アルゴリズム 3.5: BoilSchreierGenerators

データデータ：有限集合Xに作用する群G、部分底$B = (\alpha_1 , \dots , \alpha_k)$

、Gの対応する部分強生成集合S、整数$1 \leqq m \leqq k$。

結果より小さいGの部分強生成集合

開始

  for $i := 1$ *to* m do

    $T := S^{i-1}$

    各$g \in T$ do

      h（T）ごとに

        if $\alpha_i^g = \alpha_i^h \neq \alpha_i$ then

          $S := (S \setminus \{h\}) \cup gh^{-1}\}$

        終了

      終了

    終わり

  終了

  S

エンド

アルゴリズム 3.6：ComputeBSGS

データ：有限集合Xに作用する群$G = \langle S \rangle$。

結果Gの基底集合と強い生成集合

.

  (base, sgs) := GetPartialBSGS(S, $\emptyset$)

  for $i := 1$ *to* |base| do

    (base, sgs) := Schreier(base, sgs, i)

    sgs := BoilSchreierGenerators(base, sgs, i)

  終わり

  リターン (ベース, *sgs*)

エンド

6. 会員テスト

$_{1n}$もちろん、暗黙の前提は、ある大きな群Hに対して$G \leqq H$であり、かつ$g \leqq H$であることであるが、我々は行列群に興味があるので、Hがある一般的な線形群であれば、これは常に真である。

このアルゴリズムは、後で見るように、定理3.6とともにシュライア

ー・シムス アルゴリズムで使われる。

　第2節で説明したように、$g \in G$であれば、gをコセット表現$g = u\ u_{nn-1}$ - - u_1 の積として因数分解することができ、u_i はG^{i-1} におけるG^i gの代表である、

アルゴリズム3.7：シュライヤー

データデータ：有限集合Xに作用する群G、部分底$B = (\alpha_1, \ldots, \alpha_k)$

および対応するGの部分強生成集合S、$j = 0, \ldots$ に対して$G^j =$

H^j となるような整数$1 \leq i \leq k$。, i - 1.

結果：可能な拡張部分基底$B = (\alpha_1, \ldots, \alpha_m)$ と、対応するGの

部分強生成S集合は、$G^j = H^j$ for $j = 0, \ldots, i$.

/*を返す関数NewBasePoint(g)が存在すると仮定する。
$p^g \neq p$ となる点 $p \in X$ */

開始

$T := S^{i-1}$

tree := ComputeSchreierTree($T, \alpha)_i$

foreach $p \in \alpha_i^{H^{i-1}}$ do

各$s \in T$ do

gen := GetSchreierGenerator(tree, p, s)

if $gen \neq 1$ then

$S :=$ S∪gen, gen^{-1} }

if $B^{gen} = B$ then

point := NewBasePoint(gen)

$B :=$ B∪ {point}

終わり

終了

終了

終了

リターン (B, S)

エンド

各$i = 1$，．nについてシュライア木T_i $\alpha_i^{G^{i-1}}$の場合、次のようになる。アルゴリズム3.2を使用して、各コセットの代表を計算する。

より具体的には、$g \in G$の場合、$g = g_1 t(\alpha^g)$ ここで、前述と同様、$t(p)$は点pに関するアルゴリズム3.2の出力であり、$g_1 \in G^1$ である。一方、$g \notin G$ であれば、$\alpha^g \notin \alpha^G$ か、$g_1 \notin G_1$ のどちらかである。したがって、$g \in G$かどうかを調べるには、帰納的に進めばよい、

そして、まずα_1^g が α_1^G であるかどうかをチェックする。もしそれが真なら、$g_1 \in G_1$ かどうかをテストする。

アルゴリズム3.8で定式化されている。

ここでは、residanceと*level*という用語を、明白な意味とともに紹介す

る。見ての通り、Schreier-Simsアルゴリズムで必要とされるように、このアルゴリズ ムは失敗したレベルを返す。n*個の*水準がすべてパスされたとしても、3.8行目で残基$r \neq 1$となることがあり、これは$g \notin G$であることを示す。

文献では、アルゴリズム3.8は通常、グループ要素g*のふるい分け*または*ストリッピング*と呼ばれている。

7. 主なアルゴリズム

最後に、Schreier-Simsアルゴリズムそのものを紹介しよう。このアルゴリズムは、アルゴリズム3.8を利用することで、考慮するSchreierジェネレーターの数を減らす、より効率的な方法を用いている。

アルゴリズム 3.8：メンバーシップ

データデータ：有限集合Xに作用する群G、基底$B = (\alpha_1, \ldots, \alpha_n)$、a

シュライアー 軌道$\alpha\, i^G$ i+1 各 $i = 0, .n$ - 1、そして

の木 T_i

グループ要素 g。

結果残滓rと脱落レベル1≦l≦n＋1。

開始
$r := g$
for $i := 1$ *to* n do
if $\alpha_i^r \not\in T_{i-1}$ then
リターン (r, i)
終わり
element := `OrbitElement`(T_{i-1}, α_i^r)
$r := r$ - 要素$^{-1}$
終了
リターン $(r, n + 1)$
エンド

定理3.6の表記法を用いて、H^i αi+1 $= H^{i+1}$ 。

$^{ii+1}$もしi = n- 1, ..., 0 instead of the other way, then for $H^n = 1$ we obviously already have a base and strong generating set, so we can use Algorithm 3.8 to check if the Schreier generators for H^{n-1} are in H^n .

H^{n-1} のすべてのシュライヤー生成集合をチェックしたとき、$H^{n-1}_{\alpha n} = H^n$ なので、定理3.6によって、H^{n-1} の基底集合と強生成集合を持つことになる。これをアルゴリズム3.9とアルゴリズム3.10に示す。

アルゴリズム 3.9: ComputeBSGS

データ：有限集合Xに作用する群$G = \langle S \rangle$。

結果Gの基底集合と強い生成集合

.
(base, sgs) := `GetPartialBSGS`$(S, \emptyset)$
for i := |base| *to* 1 do
(base, sgs) := `SchreierSims`(base, sgs, i)
終了
リターン (ベース, *sgs*)
エンド

7.1. 行列群。与えられたアルゴリズムでわかるように、Schreier-Simsアルゴリズムの主要部分は、群の特定のタイプに依存しないが、我々は、$G \leqq \mathrm{GL}(d, q)$で、ある$d \geqq 1$、ある$q = p^r$ (pは素数、$r \geqq 1$)の状況に興味がある。ここで、*dはGの次数と*呼ばれ、*Gの*行列の行（列）の数であり、qは有限体サイズ、つまり*Gは*$\mathrm{GF}(q) = \mathbb{F}_q$ 上の行列を含んでいる。

アルゴリズム 3.10: SchreierSims

データデータ：有限集合Xに作用する群G、Gの部分基底$B = (\alpha_1, \ldots, \alpha_k)$お���������� する部分強生成集合S��$H^{j-1} = H^j$ for j = i + 1, ... , kのような整数$1 \le i \le k$、および$j = i + 1, \ldots, k$のためのα j-1のためのシュライア木T。

k、および $j = i + 1, \ldots, k$ の $\alpha^{H^{j-1}}$ のシュライア木 T^{j-1}。, k.

結果拡張された部分基底$B = (\alpha_1, \ldots, \alpha_m)$ と、$H^{j-1}_{\alpha_j} = H^j$ のようなGの対応する部分強生成S集合。

$j = i, .m$、シュライアーの木 T^{j-1} for $\alpha_j^{H^{j-1}}$ for $j = i, \ldots, m$.

/* $p^g \neq p$ となる点$p \in X$を返す関数NewBasePoint(g)が存在すると仮定する。 */

開始
    gens := S^i
    T^{i-1} := ComputeSchreierTree(gens, α_i)
    foreach $p \in \alpha_i^{H^{i-1}}$ do
        各 $s \in$ *gens* do
            gen := GetSchreierGenerator(T^{i-1}, p, s)
            if *gen* /= 1 then
                (residue, dropout) := Membership(T^i, ... , T^k, gen)
                if *residue* /= 1 then
                    S := S∪gen, gen^{-1} }
                    もし*ドロップアウト*＝k＋1なら
                        point := NewBasePoint(gen)
                        B := B∪ {point}
                    終了
                    for j := r *to* $i + 1$ do
                        (B, S) := SchreierSims(B, S, j)
                    終了
                終了
            終了
        終了
    終了
    リターン (B, S)
エンド

ベクトル空間 $X = F^d$ 上の行ベクトルと行列の乗算。F^d の標準底を e_1, e_2,
q

... で表す。e_d .

残るは、NewBasePointというアルゴリズムである。

特定の点集合Xと使用されるアクション。このアルゴリズムを構築するために、行列$M \neq I \in G$ が与えられたとき、もし$M_{ij} \neq 0$ for some $i \neq j$ ならば、$e_i M \neq e_i$ したがって、行ベクトルe_i はMによって移動された点であることを観察する。Mが対角行列であるがスカラー行列でない場合、$M_{ii} \neq M_{jj}$ for some $i \neq j$, and $(e_i + e_j)M \neq e_i + e_j$ したがって、この行ベクトルはMによって移動される。最後に，Mがスカラー行列の場合，$M \neq I$なので，$M_{11} \neq 1$となり，$e_1 M \neq e_1$ となる． これは，アルゴリズム3.11に直接変換される．

アルゴリズム 3.11: `NewBasePoint`

データ行列 $M \neq I \in G \leq \mathrm{GL}(d, q)$.

結果行ベクトル$v \in F^d$ 、$vM \neq v$となる。

```
.
for i := 1 to d do
    for j := 1 to d do
        if i /= j and M_ij /= 0 then
            e_i
        終わり
    終了
終了
for i := 1 to d do
    for j := 1 to d do
        if i /= j and M_ii /= M_jj then
            return e_i + e_j
        終了
    終了
終了
e_1
エンド
```

8. 複雑性解析

ここで、与えられたアルゴリズムの時間複雑性を解析するが、空間複雑性には興味がない。解析のために、使用する特定のデータ構造に依存する外部関数`Tree`、`AddChild`、`EdgeLabel`に$\Theta(1)$の時間がかかると仮定する。これは妥当な仮定であり、ハッシュテーブルを使ってシュライアー木を実装するプロジェクトのコードでは満たされている。また、集合に使用されるデータ構造はソートされたリストであり、バイナリサーチを使用して対数時間で要素の発見、追加、削除が可能であると仮定する。この仮定はGAPでは満たされている。

さらに、ベクトル空間$X = F^d$ に作用する行列群$G = \langle S \rangle \leq \mathrm{GL}(d, q)$で作業しているので、$F_q$ からの2つの要素の乗算に$\Theta(1)$時間がかかるという仮定の下で、2つの群要素の乗算に$\Theta(d^3)$時間がかかり、ある点への群要素の作用に$\Theta(d^2)$時間がかかることが分かっている。この仮定は、qが大きすぎず、1つのフィールド要素を機械語に格納し、一定時間で操作で

きる場合に興味があるため、妥当である。また、2つのグループ要素間の等質性の検定には$O(d^2)$の時間がかかり、2点間の等質性の検定には$O(d)$の時間がかかることがわかる。

まずアルゴリズム3.1を考えてみよう。これは幅優先探索であり、グラフ$G = (V, E)$の単純な幅優先探索には$O(|V| + |E|)$の時間がかかることが分かっている。我々の場合、｜E｜＝ /V｜ ｜S｜＝｜X｜ ｜S｜であるから、辺が支配的である。また、3.1行目は$\Theta(d^2)$の時間がかかり、3.1行目は$O(\log |S|)$の時間がかかることがわかる。したがって、前者の行が支配的であり、アルゴリズム3.1が$O(|X|\,|S|\,d^2)$の時間を要することがわかる。

アルゴリズム3.2については、頂点集合Vを持つ木において、任意のノードの深さは｜V｜で境界づけられることに注意すればよい。我々の場合、シュライヤー木の頂点は軌道の点であり、作用が推移的であればX全体である。したがって、アルゴリズム3.2は時間$O(|X|\ d^4)$を要することがわかる。

アルゴリズム3.11が時間$O(d^3)$を要することは明らかであり、アルゴリズム3.4が時間$O(|X|\ d^2 + d^3) = O(|X|\ d^2)$を要することも明らかである。アルゴリズム3.3では、3.3行目に$O(1)$の時間がかかることがわかる。なぜなら、ベースは単なるリストであり、したがって定数時間で拡張できるからである。3.3行目には$O(|B|\ d^3)$の時間がかかることがわかる。これは、実際に基底を固定する場合に、すべての基底点にグループ要素を掛け合わせ、等式をチェックしなければならない可能性があるからである。したがって、アルゴリズム3.3は時間$O(|S| \log |S|\ |B|\ d^3)$を要する。

アルゴリズム3.7はプロジェクトのGAPパッケージでは使用されないので、このアルゴリズムとその関連アルゴリズムの複雑さ解析は省略する。また、Schreier-Simsアルゴリズム自体の複雑さの詳細な解析も省略する。行列群については、いずれにせよ無駄であることがわかるからである。順列群の場合は[But91]で解析されており、その結果、Schreier-Simsアルゴリズムの時間複雑度は$O(|X|\ )$である。[5]

行列群に対するアルゴリズムを使用する場合の本質的な問題は、次のようなものである。

$X = \mathrm{F}^d_q$ したがって ／X｜ $= q^d$ 、Schreier-Simsアルゴリズムはdに指数関数的である。qが大きい場合よりもdが大きくなる場合の方が我々の関心事であるため、これは非常に残念なことである。しかし、多くの実用的な目的には十分高速な実装が可能なので、すべてが失われたわけではない。

第4章

確率的Schreier-Simsアルゴリズム

確率論的アルゴリズムは、同じ問題に対して、対応する決定論的アルゴリズムよりもはるかに単純な構造であることがよく知られている。アルゴリズムが単純であるということは、通常、より効率的で実装が容易であることを意味する。

一方、確率論的アプローチの欠点は、ある意味でアルゴリズムが「嘘をつく」、つまり正しい解を返さない可能性がゼロではないことである。このテーマと様々な複雑さのクラスについての紹介は[Pap94]にある。

確率的シュライアー・シムズアルゴリズムは、このような状況の良い例である。これは[Leo80]で初めて記述されたもので、そのアイデアは以下から来ている。

THEOREM 4.1. *Let G be a group acting on the finite set X, and let* $B = (\alpha_1, \ldots, \alpha_k)$ *be a partial base and S a partial strong generating set for G. For* $i = 1, \ldots n$, $G^i = G_{\alpha_1,\ldots,\alpha_i}$, $S^i = S \cap G^i$, $H^i = \langle S^i \rangle$, $G = G^0 = H^0$ *とする。* T^i *は、* H^{i-1} *における* H^{i-1} *のコセットの右横線である。* $|G| = \prod_{i=1}^{n} T^i$.

証明しよう。$T^i = [H^{i-1} : H^{i-1}_{\alpha_i}]$であるから、以下が成り立つ。

$$|G| = \prod_{i=1}^{n} [H^{i-1} : H^i] = \prod_{i=1}^{n} [H^{i-1} : H^{i-1}_{\alpha_i}][H^{i-1}_{\alpha_i} : H^i] = \tag{0.1}$$
$$= \prod_{i=1}^{n} T^i \prod_{i=1}^{n} [H^{i-1}_{\alpha_i} : H^i]。$$

したがって定理はこうなる。

4.2節*Gを有限集合Xに作用する群とし、*$B = (\alpha_1, \ldots, \alpha_k)$ *を部分基底、SをGの部分強生成集合とする。BとSが完全でなく、g∈Gが一様にランダムな要素である場合、gが与えられたときにアルゴリズム3.8が残基r* $\neq 1$*を返す確率は少なくとも1/2である。*

証明しよう。BとSが完全でなくても、シュライアーを計算できることがわかる。

軌道α^{H} iの木 $_{i+1}$ を $i = 0, .n - 1$ とし、それらをアルゴリズム3.8に送る。
そして、$[H^{i-1}$ $]$の対応する右横線T^i、gを表現しようとする：
こんにちは$_{\alpha i}$-1]。

したがって、アルゴリズム3.8は、$g\in\prod_{i=1}^{n} T^i$、定理4.1により$\prod_{i=1}^{n} T^i$ が

したがって、Gの要素の最大半分を含むことになる。

gは一様ランダムであり、$\Pr[g\notin\prod_{i=1}^{n} T^i] \geq 1/2$ である。

これは、基底と強生成集合を計算するための確率的アルゴリズムを示唆している。Gの部分基底と部分強生成集合が与えられたとき、次のように計算する。

各要素に対してアルゴリズム3.8を使用し、もしそれが自明でない残差を返したら、それを部分SGSに追加し、場合によっては基底を増強する。

基底とSGSが完全であれば、もちろん残基はトリビアルである。一方、基底とSGSが完全でない場合、*k個の*連続するランダム要素が三値残基を持つ確率は2^{-k}より小さい。したがって、目的のために十分な大きさの*kを*選び、*k個の*連続するランダム要素を恒等式まで取り除いたとき、基底とSGSが完全であると仮定することができる。これはアルゴリズム4.1で定式化されている。

アルゴリズム 4.1：RandomSchreierSims

データデータ：有限集合Xに作用する群G、部分底$B = (\alpha_1, \ldots, \alpha_k)$および$G$の対応する部分強生$_i$成集合$S$、整数$m \geq 1$、および$i = 1, \ldots,$ kの$\alpha^{H^{i-1}の}$シュライア木T^{i-1} 。, k.

結果：可能性のある拡張部分基底$B = (\alpha_1, \ldots, \alpha_n)$ と、対応する*Gの*部分強生成*Sセットは*、*m個の*連続するランダム要素がBと*S に関して*同一になるように取り除かれている。

/*を返す関数NewBasePoint(g)が存在すると仮定する。
p^g /=pとなるような点$p \in X$。 */

/*グループ*Gから*一様にランダムな要素を返す関数Random(G)の存在を仮定する。 */

開始
  sifts := 0
  while *sifts* < m do
    element := Random(G)
    (residue, dropout) := Membership($T^1, \ldots, T^k$, element)
    if *residue* /= 1 then
      S := S∪ element, element^{-1}}
      dropout = $k + 1$ ならば
        point := NewBasePoint(要素)
        B := B∪ {point}
        $k := k + 1$
      終わり
      for i := 1 *to* k do
        T^i := ComputeSchreierTree(S^i, $\alpha)_i$
      終了
      sifts := 0
    その他
      sifts := sifts + 1

```
        終了
    終了
    リターン (B, S)
エンド
```

実際、アルゴリズム4.1はアルゴリズム3.10よりも単純であり、実装は通常より高速である。しかし、Randomアルゴリズムがどのように構成されるべきかは明確ではない。まず、ランダムビットが必要であり、擬似ランダムビットの生成はコンピュータサイエンスの古い問題である[Pap94]や[Knu97]を参照されたい。その代わりに、一様なランダムビットが利用可能であると仮定する。

1. ランダムなグループ要素

ランダムな群要素の生成も一般に難しく、このトピックに関する入門書は[Bab97]にある。もし基底と強い生成集合を知っていれば、セクション2で説明した因数分解を使用して、各横軸からランダムな要素を選択し、それらを乗算することで簡単にランダムな要素を生成することができます。そして、（ほぼ）一様にランダムな群要素を*証明的に*生成できる唯一の既知のアルゴリズムは[Bab91]に記述されており、GL(d, q)の部分群に対して時間O(d^{10} (log q)^5)で実行され、これは我々の目的には遅すぎます。

実用的なアルゴリズムは、[ACG$^+$ 99]を参照の*ヒューリスティックス*であり、一様なランダム性の保証は*証明され*ていない。最も成功したアルゴリズムは、"Shake "とも呼ばれる*積の置換*アルゴリズムであり、これは元々Charles Leedham-GreenとLeonard Soicherによるアイデアであり、[CLGM$^+$ 95]で説明されている。

このアルゴリズ ムの分析には深入りしないが、アルゴリズム自体は非常に単純である。グループ$G = \langle g_1 , , g_n \rangle$ が与えられたとき、"Shake "アルゴリズムは主に次のようになる。

はグローバル変数$S = (a_1 , , a_m) \in G^m$ for some $m \geq n$.を保持し、各呼び出しは次のようになる。

をアルゴリズム4.2に適用すると、提案されたランダムなグループ要素が返される。

アルゴリズム4.2：シェイク

データ：グローバル状態$S = (a_1 , \dots , a_m) \in G^m$

結果：Gの一様ランダム要素の良い近似であることを望むGの要素。

/*を返す関数RandomInteger(k)が存在すると仮定する。
k}の集合{0, ., k} */

開始

i := RandomInteger(m)

繰り返す
    j := `RandomInteger`(m)
i /= j まで
もし*RandomInteger*(1) = 0なら
    $b := a\ a_{ij}$
その他
    $b := a\ a_{ji}$
終了
$a_i := b$
戻り b
エンド

明らかに、このアルゴリズムは時間的にも空間的にも非常に安価である。残された問題は、状態Sをどのように初期化し、mをどのように選択するかであり、一様にランダムなグループ要素を生成するアルゴリズムの能力は、その答えに大きく依存する。CLGM+ 95]では、$m = \max(10, 2n + 1)$��し、Sは生成子$g_1, \ldots, g$を含むように初期化することを提案している。g_nを含むように初期化される。さらに、状態はアルゴリズム4.2をK回呼び出し、結果を破棄することで初期化される。

NP01]では、このアルゴリズムと、Leedham-Greenによる "Rattle "と呼ばれるその変種も記述されている。両アルゴリズムは比較され、"Rattle "の方がわずかに優れているが、実行時間は少し長い。GAPには、このプロジェクトで使用する "Shake "アルゴリズムの実装がある。Rattle "の実装も行われたが、十分な効率が得られないことが判明した。

第5章

Schreier-Todd-Coxeter-Simsアルゴリズム

コセット列挙は群論で最も古いアルゴリズムの一つであり、[TC36]で初めて記述された。有限体表現群Gと$H \leqq G$が与えられたとき、コセット列挙の目的は、GにおけるHのコセットのリストを構築すること、すなわち、GがXに推移的に作用し、$H = G_p$ となるような点$p \in X$を持つ集合Xを見つけることである。$n\}$、$p = 1$である。Hのインデックスが有限でない場合、アルゴリズムは通常終了しない。

コセットの列挙は試行錯誤のプロセスであり、かなりの数の戦略が長年にわたって開発されてきた。実際のアルゴリズムについては、このレポートの範囲外であり、このプロジェクトで使用されているGAPに優れた実装があるので、ここでは触れない。コセット列挙に関するより詳細な説明は[Sim03]にある。

しかし、コセット列挙の全体的な構造を知っておくことは重要である。Gの要素をXに作用させ、必要に応じて新しいコセットを導入する。そして、いくつかのコセットが同じであることが判明し、リストで識別される。このアルゴリズムの構造は、与えられた数のコセットが定義されたときに、早期に終了することを可能にする。

我々の場合、定理3.6の最後のケースを使いたい。Schreier-Simsアルゴリズムのレベルiでは、ベースとSGSはより高いレベルに対して完全であることが分かっている、

そして、$H^{i+1} = H$であることを$_{\alpha i+1}$ と等価である。

検証したい。^{i}

$[H^i : H^{i+1}] = \alpha^H$ i であり、α^H i のシュライアツリーを計算できる。 存じております

i+1 i+1

軌道サイズ。したがって、コセット列挙を使って$[H^i : H^{i+1}]$を計算し、それが軌道サイズと等しいことがわかれば、シュライヤー生成子を計算してメンバーシップをチェックする必要はない。

このすべてはアルゴリズム5.1で定式化されている。

$M \alpha^H$ i コセットが定義された後の列挙は、ある有理数について

i+
1
$M \geqq 1$、これは確かに上限だからである。我々の実装では、数M*は*ユーザーが指定でき、デフォルトは$M = 6/5$である。この値は[Leo80]に由来しており、そこではアルゴリズムが紹介され、M*の*良い値を決定するためにいくつかの実験が行われた。

Schreier-Todd-Coxeter-SimsアルゴリズムはS、入力部分ベースとSGSが既に完成している場合に、特に優れた性能を発揮することが知られている。したがって、確率的アルゴリズムからの出力の*検証に*使用することができ、このプロジェクトでは主にそのように使用し、前述の確率的アルゴリズムからの出力を検証する。

アルゴリズム5.1：SchreierToddCoxeterSims

データデータ：有限集合Xに作用する群G、Gの部分基底$B = (\alpha_1, \ldots, \alpha_k)$お�����������する部分強生成集合S��$H^{j-1} = H^j$ for j = i + 1, ... , kのような整数$1 \le i \le k$、および$j = i + 1, \ldots, k$のためのα j-1のためのシュライア木T。

k、および $j = i + 1, \ldots, k$ の $\alpha^{H^{j-1}}$ のシュライア木 T^{j-1} 。, k.

結果拡張された部分基底$B = (\alpha_1, \ldots, \alpha_m)$ と、$H^{j-1}_{\alpha_j} = H^j$ のようなGの対応する部分強生成S集合。

$j = i, .m$、シュライアーの木 T^{j-1} for $\alpha_j^{H^{j-1}}$ for $j = i, . ., m$.

/*p^g /= pとなる点$p \in X$を返す関数NewBasePoint(g)の存在を仮定する。 */

/*以下の関数ToddCoxeter(U_1, U_2, k)が存在すると仮定する。
$G = \langle U_1 \rangle$ と $H = \rangle U_2 \rangle \le G$ のコセット列挙を行い、k個のコセットが定義されたら終了する。 */

```
開始
    gens := S^i
    T^{i−1} := ComputeSchreierTree(gens, α_i)
    foreach p ∈ α_i^{H^{i-1}} do
        各 s ∈ gens do
            table := ToddCoxeter(S^i, S^{i+1}, |T^i| + 1)
            if |table| = |T^i| then
                リターン (B, S)
            終了
            gen := GetSchreierGenerator(T^{i−1}, p, s)
            if gen /= 1 then
                (residue, dropout) := Membership(T^i, ... , T^k, gen)
                if residue /= 1 then
                    S := S∪gen, gen^{−1}   }
                    もしドロップアウト＝k＋1なら
                        point := NewBasePoint(gen)
                        B := B∪ {point}
                    終了
                    for j := r to i + 1 do
                        (B, S) := SchreierToddCoxeterSims(B, S, j)
                    終了
                終了
            終了
        終了
```

終了

戻り (B, S)

エンド

第6章

実装と最適化

ここからは、プロジェクトにおけるコードの実際の実装と最適化に関する、より実際的な問題について考えていく。

GAPでは、Schreier-Simsアルゴリズムの非常に高速な実装がすでに存在する。これは[Ser03]に記述されており、[BCFA´S91]に記述されているアルゴリズムに基づいた発見的アルゴリズムである。GAPの他の多くのアルゴリズムと同様に、このアルゴリズムは変異群でのみ動作するため、一般的な群Gに対して、まず忠実な順列表現、つまり、Cayleyによるよく知られた定理により存在する有限集合Xに対する射影同型性$\lambda : G \to \mathrm{Sym}(X)$を計算する。次に
のイメージ$\lambda(G)$、あるいはむしろ$H \sim= \lambda(G)$となるような$H \leq S_{|X|}$。

このプロジェクトで開発されたGAPパッケージの最も重要な動機は、もしGが行列群であり、最初に順列に変換する代わりに行列を直接扱うのであれば、そうでなければ捨てられてしまう付加的な構造を利用し、より高速なアルゴリズムを思いつくことができるかもしれないという考えである。したがって、このコードは、すべての群要素が有限体上の可逆行列であるという基本的な前提で作成され、そのような行列の内部GAP表現を使用する。

1. シュライヤーの木

重要な問題は、シュライアーの木をどのように管理するかである。実際に我々が本当に欲しいのは、シュライヤーの木によって定義された横木である。したがって、可能性としては、木ではなく、すべての点のリストと、それぞれの点に対して、ルートをこの点に移動させる要素を保存することである。これは高さ1の木として実現することができ、辺のラベルはシュライアーの木の場合のように生成子ではなく、グループの要素である。一方、シュライアー木でコセット代表を見つけるには、ある点からルートへのパスをたどる必要があり、平均して対数時間がかかり

、木のバランスが適切でない場合は線形時間がかかるかもしれない。

結論として、我々は時間と空間の間でトレードオフの関係にあり、このプロジェクトでは、上記の両方の戦略をパッケージのユーザーが利用できる。ツリーの具体的な表現はハッシュ・テーブルであり、キーは点、値はルートに向かうユニークなエッジのエッジ・ラベルである。これにより、与えられた点がシュライアーツリーで定義された軌道上にあるかどうかをチェックするという一般的なタスクを定数時間で実行することが可能になる。

1.1. 拡張と作成。アルゴリズム3.10をより詳細に考えてみると、集合S^i は変更された場合のみ拡張されることに気づく。したがって

3.10行目で、計算されたツリーを保存しておき、次にそこに到着したときに、新しいジェネレーターがあれば、それを使ってツリーを拡張するだけにすることもできる。

これは、より高速なアルゴリズムを与えるかもしれないし、与えないかもしれない。明らかに、アルゴリズム3.1では、シュライア ツリーを拡張する方が、新しいツリーを作るよりも少ない作業で済む。木が釣り合っていない場合、アルゴリズム3.2はより多くの時間を要し、実際、この関数に最も多くの時間が費やされる。Ser03]では、このような木がアンバランスな問題は行列群ではより一般的であると主張されており、このプロジェクトでの経験的研究により、ほとんどの場合、毎回シュライアー木を再計算した方がよいことが示されている。

1.2. 浅い木。Schreier木を作成するためのアルゴリズムがあり、それはSchreier木が*浅く*、つまりバランスが取れていて、最悪対数の高さになるように保証されている。Ser03]には、決定論的アルゴリズムと確率論的アルゴリズムの2つが記述されている。このプロジェクトでは、決定論的アルゴリズムは
BCFA´S91]にも記述されている。これは複雑すぎて
ここではより詳しく説明するが、本質的なアイデアは、与えられたジェネレーターではなく、異なるエッジラベルのセットを選択することである。

2. 軌道サイズ

先に述べたように、行列群$G \leqq \mathrm{GL}(d, q)$に対してSchreier-Simsアルゴリズムを使うことは、ある意味で最初から運命づけられている。なぜなら、Gが$\mathbb{F}_q^d$ に作用するとき、その複雑さはdの指数関数になるからである。この場合の運命の現れの一つは、軌道が巨大になる可能性があることで、これは順列群では起こらないことである。このため、シュライヤー木が巨大になり、大量のシュライヤー生成を作らなければならないので、アルゴリズムが遅くなる。

2.1. 交互作用。大きな軌道を避けるために、Gの別の作用を使うことができる。しかし、その作用が忠実である場合のみ、Gの順列表現となり、Schreier-Simsアルゴリズムが機能するためにはこれが必要である。

But76]では、基点が1次元部分空間（線）とその線からのベクトルとして交互に選択されるという巧妙なトリックが紹介された。ベクトル$v = (v_1, \ldots, v_d) \in \mathrm{F}^d$ が基点として選ばれたとき、その基点には*線分*〉v ↪Pe_27E8 *が*先行する。線上のG*の*作用は*射影作用として*知られ、もちろん忠実ではないが、次の点vはカーネルのものなので、これは忠実ではない。
の問題である。また、部分空間↪Ps_27E8 は正準代表（$1, v\, v_2^{-1}, \ldots, v\, v_d^{-1}$）を持つ、従って、射影作用は、時間の観点から次のようになる。
ポイントのアクションより特に高いわけではない。

Now, if $k = v^G$ then $\langle v\rangle^G$ $v^{G\langle v\rangle} = k$ and $G_{\langle v\rangle}$ is the stabiliser of the line

このことは、$u \in v^{G}$ (v↪Pe_27E9) なら u もその線上にあることを意味するので、$u = mv$ となる。

ここで、$m \in \mathrm{F}_q$。$M = \mathrm{m} \in \mathrm{F}_q \mid mv \in v^{G}$ (v)) であることがわかる。はF* の部分群であり

thus $l = v^{G\langle v\rangle} = |M|$ divides $q - 1$.

したがって、大きさkの1つの軌道の代わりに、大きさk/lとlの2つの軌道を持つことになり、$l \geqq 2$であればいつでも$k \geqq k/l + l$ *である*ため、例えば、計算とふるい分けに必要なシュライヤー生成数が少なくなる。その一方で、我々のベースはおそらく長くなるだろう

このトリックを使用する場合、必ずしも良いアイデアとは限らないが、それでも実装され、使用できることが、このプロジェクトにおける実証的研究で示されている。

2.2.固有空間。より小さな軌道を生成するもう1つの方法が[MO95]に記述されている。そのアイデアは、与えられた生成行列の固有ベクトルとなる基点を選ぶことです。

行列$A \in \mathrm{GL}(d, q)$の*特性多項式は*$c_A(x) = \det(xI - A)$であり、Iは恒等行列であり、A*の固有値は*$c_A(x)$の根であることを線形代数から思い出し$_q$てください。通常、固有ベクトルは $v^A = \lambda v$ (λ*は*A*の*固有値)、または $v^{g(A)} = 0$ ($g(x)$は$c_A(x)$の線形因子)のようなベクトル$v \in \mathrm{F}^d$ として定義される。ここでは、後者のより一般的な定義を用いる、
ここで、$c_A(x)$の線形因子だけでなく、高次の因子も許容する。

$^{\langle A \rangle \alpha A *}$線形因子 $g(x) = x - \lambda$ および対応する固有ベクトル v に対して、軌道 $v^{\langle A \rangle}$ の大きさは $q - 1$ の約数である。

q

$c_A(x)$の次数m*の*因子である場合、固有ベクトルv*の*軌道$v^{\langle A \rangle}$ の大きさは、 $q^m - 1$で上界される。したがって、可能な限り小さな軌道を得るためには、できるだけ次数の小さい因数に対応する固有ベクトルを選ぶべきである。

行列群$G =$ ⟨S↪Pe_27E9 の軌道は、S中の生成子の一つの固有ベクトルを基点として選んだからといって小さくなる必要はないが、いくつかの生成子の固有ベクトルである基点を選ぶと、軌道の大きさが小さくなることが多いことに注意されたい。MO95]では、これらの問題が詳細に調査され、実験され、うまくいけば小さな軌道を与える基点を見つけるための発見的な方法が開発された。これはGAPにも実装されており、このプロジェクトでもそのアルゴリズムを使うことが可能である。しかし、オーバーヘッドがあり、軌道サイズが実際に小さくなるかどうか定かでないため、これを使うのは必ずしも良いアイデアとは言えない。

3. さらなる発展

このパッケージには、提案されたベースとSGSが完全かどうかを検証するアルゴリズ ムである、Simsによるいわゆる*Verify*ルーチンと、ベースとSGSを見つけるための ほぼ線形時間のアルゴリズムという、2つのより高度なアルゴリズムの実装が 含まれている。これらのアルゴリズムの

完全な説明は本レポートの範囲を超えているが、完全を期すために、これらに関する簡単な説明を含む。

3.1. Verifyルーチン。このアルゴリズムはSimsによるもので、出版されたことはないが、[Ser03]に記述されている。有限集合Xに作用する群G = ⟨S⟨、点$\alpha \in X$、および部分群H = ⟨S' ⟩ ≤ G_α が与えられたとき、$H = G_\alpha$かどうかをチェックする。 もしそうでない場合、アルゴリズムは$g \in G_\alpha$ ⟨ Hを計算してそれを証明する。提案された基底全体とSGSをチェックするには、定理3.6の3番目のケースを使用し、 各レベルをチェックする。

このアルゴリズムは、理論的にも、実装する際にも、かなり複雑である。再帰的な性質があり、S(′)上で帰納され、基点の変更やブロックシステムの計算などが含まれる。

3.2. ほぼ線形時間のアルゴリズムX上で作用する順列群G*に対し*て、ほぼ線形時間、すなわち｜X｜と｜G｜の対数因子を除いた｜X /*に*線形な時間で実行される基底とSGSを計算するアルゴリズムがある。これは時間の複雑さの点で最もよく知られたアルゴリズムであり、[BCFA´S91]に記述されている。
Ser03]と同様に確率的である。この複雑さは、浅いSchreier木、高速な確率的検証アルゴリズム、 およびランダムに選択された数個のSchreierジェネレータのみをふるいにかける ことによって達成される。それ以外の点では、このアルゴリズムは、先に説明したベースとSGSを計算する アルゴリズムとよく似ている。

ランダムなSchreierジェネレータは、[Ser03]で説明されている *ランダムな部分*積とランダムなグループ要素を選択することによって計算され、アルゴリズムは、計算するSchreierジェネレータの数と与えられた正しさの確率に関連するいくつかの定理に依存している。

第7章

パフォーマンス

先に述べたように、このプロジェクトの目的のひとつは、GAPにすでに存在するものよりも高速な実装を行うことであった。この目的が達成されたかどうかを判断するために、この実装はベンチマークされ、GAPに組み込まれている実装と比較された。

このアルゴリズムは、GAPで簡単に構成できるいくつかの行列群の基底とSGSを計算するために使用され、使用された生成集合はGAPライブラリの標準生成集合であった。主なテスト群は古典群であり、様々な（小さな）dと*qに対し*て、一般的で特殊な線形群GL(d, q)とSL(d, q)、一般的で特殊な直交群GO(d, q)とSO(d, q)であった。また、いくつかのSuzuki群Sz(q)（ここでqは2の非2乗）といくつかのRee群Ree(q)（ここで$q = 3^{1+2m}$、いくつかの$m > 0$）に対してもアルゴリズムをテストした。

もう一つの、おそらくより現実的な性能テストは、与えられた有限体上の与えられた次元の可逆行列のランダムに形成された集合に対してアルゴリズムを実行することによって行われた。

最初のテストは、AMD Athlon CPUを搭載し、2GHzで動作し、1GBの物理RAMを搭載したコンピュータで、2番目のテストは、Intel Pentium 4 CPUを搭載し、2.8GHzで動作し、同じく1GBの物理RAMを搭載したコンピュータで実施した。GAPは主にCPUを集中的に使用し、ベンチマーク中はスワッピングを避けたため、ハードディスクの速度は無視できる程度のものであった。GAPのインストール・テストでは、GAP4stonesの値がそれぞれ194624と253581であった。

ベンチマークの詳細を付録に示す。最初のテストでは、ほとんどのグループで既存のアルゴリズムの方が速かったが、いくつかのグループでは我々の実装の方が速かった。注意すべきは、我々の実装のすべての実行時間は、同じアルゴリズム・オプションを使用したことによるものであり、他のオプションを工夫することにより、特に小さなグループにおいて、より良い時間を得ることが可能であることである。2回目のテスト

でも、ほとんどのケースで既存の実装の方が速いことが示された。

メモリに関しては、厳密なベンチマークは実施されていないが、上記のベンチマーク中に、GAPプロセスによって割り当てられたメモリ量の簡単なチェックがいくつか実施された。その結果、我々の実装の方が全体的にメモリ消費量が少なく、場合によってはその差が大きかった。実際、ベンチマーク中のスワッピングを避けたかったため、最初のテストではSz(32)よりも大きなSuzukiグループをチェックすることができなかった。我々の実装はそのような問題からは程遠く、鈴木グループで約50MB以上を消費することはなかった。

したがって、このプロジェクトは小さな成功を収めたという結論にならざるを得ない。

付録A

ベンチマーク

以下は最初のベンチマークの結果の詳細である。最初の列はGAPで記述されたテストグループを示し、他の列はミリ秒単位の実行時間を示す。アルゴリズムを比較するために使用された方法は、計算されたベースとSGSの軌道サイズを使用して入力グループの順序を計算することでしたので、既存の実装を測定するために、次のようなコマンドを使用しました。

```
gap> Size(Group(GeneratorsOfGroup(GL(4, 4))));
gap> benchmark_time := time;
```

表1：ベンチマーク結果

グループ	プロジェクト	ギャップ
SL(2, 2)	110	170
GO(+1, 2, 2)	20	10
GO(-1, 2, 2)	20	0
GL(2, 4)	80	20
SL(2, 4)	60	10
GO(+1, 2, 4)	40	0
GO(-1, 2, 4)	40	10
GL(2, 3)	80	30
SL(2, 3)	50	0
GO(+1, 2, 3)	50	10
GO(-1, 2, 3)	50	0
SO(+1, 2, 3)	20	0
SO(-1, 2, 3)	30	10
GL(2, 5)	70	10
SL(2, 5)	40	10
GO(+1, 2, 5)	50	10
GO(-1, 2, 5)	50	0
SO(+1, 2, 5)	50	10
SO(-1, 2, 5)	40	0
SL(3, 2)	50	10
GO(0, 3, 2)	40	0
GL(3, 4)	170	20

表2：ベンチマーク結果

グループ	プロジェクト	ギャップ
SL(3, 4)	110	20
GO(0, 3, 4)	90	10
GL(3, 3)	70	10
SL(3, 3)	100	0
GO(0, 3, 3)	60	10
SO(0, 3, 3)	50	0
GL(3, 5)	190	60
SL(3, 5)	110	40
GO(0, 3, 5)	70	10
SO(0, 3, 5)	70	20
SL(4, 2)	130	10
GO(+1, 4, 2)	40	10
GO(-1, 4, 2)	60	0
GL(4, 4)	1480	30
SL(4, 4)	690	80
GO(+1, 4, 4)	90	30
GO(-1, 4, 4)	170	20
GL(4, 3)	210	30
SL(4, 3)	410	20
GO(+1, 4, 3)	80	20
GO(-1, 4, 3)	80	10
SO(+1, 4, 3)	80	0
SO(-1, 4, 3)	80	10
GL(4, 5)	1600	50
SL(4, 5)	790	190
GO(+1, 4, 5)	100	60
GO(-1, 4, 5)	140	50
SO(+1, 4, 5)	140	60
SO(-1, 4, 5)	100	60
SL(5, 2)	560	10
GO(0, 5, 2)	90	10
GL(5, 4)	26440	220
SL(5, 4)	4650	440
GO(0, 5, 4)	230	100

表3：ベンチマーク結果

グループ	プロジェクト	ギャップ
GL(5, 3)	890	30
SL(5, 3)	5090	130
GO(0, 5, 3)	100	40
SO(0, 5, 3)	150	40
GL(5, 5)	24500	410
SL(5, 5)	12380	1450
GO(0, 5, 5)	470	370
SO(0, 5, 5)	390	380
SL(6, 2)	1250	40
GO(+1, 6, 2)	250	20
GO(-1, 6, 2)	350	10
GL(6, 4)	205430	460
SL(6, 4)	89580	4030
GO(+1, 6, 4)	3880 580	
GO(-1, 6, 4)	3700	550
GL(6, 3)	3850	100
SL(6, 3)	16890	660
GO(+1, 6, 3)	340	90
GO(-1, 6, 3)	340	200
SO(+1, 6, 3)	520	100
SO(-1, 6, 3)	480	100
GL(6, 5)	292220	84030
SL(6, 5)	227830	82010
GO(+1, 6, 5)	5410	2050
GO(-1, 6, 5)	1710	1440
SO(+1, 6, 5)	3190	1460
SO(-1, 6, 5)	3520	1390
Sz(8)	180	1170
Sz(32)	4390	133860
Sz(128)	1084510	> 3600000
リー(27)	816590	687780

以下は2番目のベンチマークの結果である。最初の列は有限体サイズ、2番目の列は行列の次元、3番目の列は形成されたランダム生成集合のサイズを示す。各生成集合は、`RandomInvertibleMat`を指定された回数呼び出すことで計算されました。表の各行について，与えられたパラメータを持つ 20 個の生成集合が形成され，これらの集合に対してアルゴリズムが実行され，これら 20 個の実行の平均時間が表の最後の 2 列に示されている．

表4: ベンチマーク結果

フィールド	薄暗い	セット	プロジェクト	ギャップ
2	2	1	25	16
2	2	2	23	4
2	2	3	23	2
2	2	4	23	2
2	2	5	23	2
2	2	6	23	2
2	2	7	23	3
2	2	8	23	2
2	2	9	24	3
2	2	10	23	3
2	3	1	34	2
2	3	2	42	3
2	3	3	49	3
2	3	4	52	3
2	3	5	53	4
2	3	6	58	4
2	3	7	60	4
2	3	8	68	4
2	3	9	62	5
2	3	10	68	4
2	4	1	46	3
2	4	2	87	4
2	4	3	117	5
2	4	4	120	6
2	4	5	126	6
2	4	6	124	7
2	4	7	145	7
2	4	8	151	7
2	4	9	161	8
2	4	10	168	8

表5: ベンチマーク結果

フィールド	薄暗い	セット	プロジェクト	ギャップ
2	5	1	58	3
2	5	2	208	10
2	5	3	231	11
2	5	4	301	12
2	5	5	294	13
2	5	6	357	14
2	5	7	384	15
2	5	8	355	15
2	5	9	407	16
2	5	10	471	17
2	6	1	75	4
2	6	2	612	24
2	6	3	530	27
2	6	4	704	28
2	6	5	1033	32
2	6	6	903	34
2	6	7	1209	37
2	6	8	1055	41
2	6	9	1300	44
2	6	10	1158	45
2	7	1	86	8
2	7	2	1229	24
2	7	3	1675	20
2	7	4	2289	20
2	7	5	2183	25
2	7	6	2404	23
2	7	7	2556	24
2	7	8	2477	25
2	7	9	2963	29
2	7	10	3343	27
4	2	1	37	3
4	2	2	55	4
4	2	3	60	5
4	2	4	61	5
4	2	5	62	5
4	2	6	64	6
4	2	7	69	7
4	2	8	70	7
4	2	9	73	7
4	2	10	75	8
4	3	1	59	5
4	3	2	118	12
4	3	3	133	14
4	3	4	152	17
4	3	5	158	19
4	3	6	175	21
4	3	7	185	22
4	3	8	215	25
4	3	9	216	27

表6: ベンチマーク結果

フィールド	薄暗い	セット	プロジェクト	ギャップ
4	4	1	75	15
4	4	2	430	26
4	4	3	576	28
4	4	4	589	29
4	4	5	749	28
4	4	6	714	31
4	4	7	838	33
4	4	8	912	38
4	4	9	1019	36
4	4	10	1085	38
4	5	1	109	52
4	5	2	3141	170
4	5	3	4248	119
4	5	4	3660	106
4	5	5	5126	116
4	5	6	3895	124
4	5	7	4416	134
4	5	8	5154	137
4	5	9	5002	147
4	5	10	6285	156
4	6	1	193	212
4	6	2	27943	655
4	6	3	33664	455
4	6	4	30882	478
4	6	5	40467	506
4	6	6	36702	534
4	6	7	35179	570
4	6	8	41211	593
4	6	9	47733	630
4	6	10	46698	661
4	7	1	8786	19238
4	7	2	335656	95870
4	7	3	337232	100999
4	7	4	371067	98525
4	7	5	381631	94616
4	7	6	463195	94353
4	7	7	463824	102045
4	7	8	460079	104262
4	7	9	485832	115177
4	7	10	497838	99240
3	2	1	39	3
3	2	2	47	3
3	2	3	49	3
3	2	4	52	3
3	2	5	51	4
3	2	6	53	4
3	2	7	46	4
3	2	8	54	4
3	2	9	54	5

4	3	10	226	28	3	2	10	47	5

表7: ベンチマーク結果

フィールド	薄暗い	セット	プロジェクト	ギャップ
3	3	1	59	4
3	3	2	92	7
3	3	3	98	8
3	3	4	101	9
3	3	5	111	9
3	3	6	114	11
3	3	7	126	12
3	3	8	126	13
3	3	9	136	12
3	3	10	143	15
3	4	1	74	7
3	4	2	244	22
3	4	3	278	26
3	4	4	310	29
3	4	5	366	32
3	4	6	334	35
3	4	7	415	39
3	4	8	452	40
3	4	9	475	45
3	4	10	454	47
3	5	1	82	15
3	5	2	1029	44
3	5	3	1110	28
3	5	4	1692	29
3	5	5	1361	36
3	5	6	1612	34
3	5	7	1752	35
3	5	8	1887	40
3	5	9	1928	39
3	5	10	2197	42
3	6	1	111	44
3	6	2	6517	105
3	6	3	5772	131
3	6	4	7463	117
3	6	5	6184	104
3	6	6	7111	128
3	6	7	7191	117
3	6	8	8478	120
3	6	9	11749	127
3	6	10	10142	134
3	7	1	385	128
3	7	2	33440	591
3	7	3	36465	586
3	7	4	35244	506
3	7	5	35492	331
3	7	6	43913	351
3	7	7	47698	364
3	7	8	45104	382
3	7	9	46772	401

表8: ベンチマーク結果

フィールド	薄暗い	セット	プロジェクト	ギャップ
5	2	1	46	3
5	2	2	59	5
5	2	3	61	6
5	2	4	64	6
5	2	5	64	8
5	2	6	65	9
5	2	7	68	9
5	2	8	73	10
5	2	9	72	10
5	2	10	77	11
5	3	1	59	8
5	3	2	147	14
5	3	3	157	15
5	3	4	178	15
5	3	5	182	14
5	3	6	203	15
5	3	7	226	17
5	3	8	240	17
5	3	9	258	19
5	3	10	269	20
5	4	1	78	35
5	4	2	957	94
5	4	3	1123	73
5	4	4	1038	65
5	4	5	1300	68
5	4	6	1337	72
5	4	7	1563	78
5	4	8	1703	81
5	4	9	1815	93
5	4	10	1864	91
5	5	1	166	158
5	5	2	10330	414
5	5	3	11236	508
5	5	4	11779	402
5	5	5	13499	355
5	5	6	15476	375
5	5	7	14578	475
5	5	8	14204	428
5	5	9	14994	450
5	5	10	15487	472
5	6	1	903	3568
5	6	2	180644	86753
5	6	3	195962	91533
5	6	4	210412	84445
5	6	5	216448	84051
5	6	6	225522	80495
5	6	7	212674	76792
5	6	8	258668	81668
5	6	9	269393	97520

3	7	10	62226	421	5	6	10	287699	87331

参考文献

[ACG+ 99] Giorgio Ausiello, Pierluigi Crescenzi, Giorgio Gambosi, Alberto Marchetti-Spaccamela, Marco Protasi, and Viggo Kann, *Complexity and Approximation : Combinatorial Optimization Problems and Their Approximability Properties*, ch. 10
、
pp.321-351, Springer, Berlin Heidelberg, 1999.

[Bab91] L´aszl´o Babai, *Local expansion of vertex-transitive graphs and random generation in groups*, Proc. 23rd ACM Symp.Theory of Computing (Los Angeles), Association for Computing Machinery, 1991, pp. 164-174.

[Bab97] ______*群アルゴリズムにおけるランダム化: グループと計算II*, アメリカ数学会DIMACSシリーズ, vol.28, 1997、
pp.1-17.

[BB96] John A. Beachy and William D. Blair, *Abstract algebra*, 2nd ed., Waveland Press, Illinois, 1996.

[BCFA´S91] L´aszl´o Babai, Gene Cooperman, Larry Finkelstein, and A´kos Seress, *Nearly linear.*
小基底を持つ並べ替え群に対する時間アルゴリズム, 記号と代数計算に関する国際シンポジウム, ISSAC '91, ACM Press, New York, 1991, pp.200-209.

[BH99] Martin R. Bridson and Andr´e Haefliger, *Metric spaces of non-positive curvature*, ch. 1, pp. 6-8, Springer, 1999.

[ビッグ89] Norman L. Biggs, *Discrete mathematics*, Oxford University Press, 1989.

[But76] Gregory Butler, *The Schreier algorithm for matrix groups*, Proc. of ACM Symposium on symbolic and algebraic computation, SYMSAC '76, Association for Computing Machinery, New York, 1976, pp.

[But91]。 ______*順列群に対する基本的アルゴリズム*、コンピュータ 科学レクチャーノート、559巻、シュプリンガー、1991年。

[CH92] John Cannon and George Havas, *Algorithms for groups*, Australian Computer Journal **24** (1992), 59-68.

[CLGM+ 95] フランク・セラー、チャールズ・R・リーダム＝グリーン、スコット・H・マレー、アリス・C・ニーマイヤー、そして
Eamonn A. O'Brien, *有限群のランダム要素の生成*, 可換 Algebra (1995), no.23, 4931-4948.

[CLR90] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest, *Introduction to algorithms*, The MIT Electrical Engineering and Computer Science Series, The MIT Press, 1990.

[CSS99] Hans Cuypers, Leonard H. Soicher, and Hans Sterk, *Working with finite groups*, Some Tapas of Computer Algebra (Arjeh M. Cohen, Hans Cuypers, and Hans Sterk, eds.), Algorithms and Computation in Mathematics, vol. 4, Springer, 1999, pp. 184-207.

[GAP] *Groups, Algorithms and Programming*, http://www.gap-system.org/

。[Hal59] Marshall Hall, *The theory of groups*, Macmillan, New York, 1959.

[Jen97] Robert J. Jenkins, *Hash functions for hash table lookup*, Dr. Dobb's Journal (1997), http://burtleburtle.net/bob/hash/evahash.html.

[Knu97] Donald E. Knuth, *The Art of Computer Programming : Seminumerical algorithms*, 第3

版, vol. 2, ch. 3, Addison-Wesley, Reading, Massachusetts, 1997.

[レオ80] ジェフリー・S・レオン(Jeffrey S. Leon)、*順列の生成によって与えられる群の基底集合と強 生成集合を求めるアルゴリズムについて*、Mathematics of Computation **35** (1980), no.151, 941-974.

[MO95] Scott H. Murray and Eamonn A. O'Brien, *Selecting base points for the Schreier-Sims algorithm for matrix groups*, Journal of Symbolic Computation **19** (1995), no. 6, 577-584.

[Mur93] Scott H. Murray, *The Schreier-Sims algorithm*, Master'ssis, Australian National University, 1993, http://www.win.tue.nl/˜smurray/research/.

[Neu95] *計算群論への招待*、Groups '93 (Gal- way/St, Andrews), London Mathematical Society Lecture Note Series, vol.212, Cam- bridge University Press, 1995, pp.457-475.

[NP01] この論文では、「論理とその応用における複雑*性と計算*」(Rod Downey and Denis Hirschfeldt, eds.), de Gruyter Series in Logic and its Applications, vol.4, de Gruyter, Berlin, 2001, pp.87-113.

[Pap94] Christos H. Papadimitriou, *Computational complexity*, 1 ed., Addison-Wesley, Janu- ary 1994.

[Ros94] John S. Rose, *A course on group theory*, Dover, 1994.

[Sch27] Otto Schreier, *Die Untergruppen der freien Gruppen*, Abh.Math.Sem.Univ. Ham-Burg **5** (1927), 161-183.

[Ser97 】。] A´kos Seress, *An introduction to computational group theory*, Notices of American Mathematical Society **44** (1997), 671-679, http://www.math.ohio-state.edu/˜akos/.

[Ser03] ______*Permutation group algorithms*, Cambridge Tracts in Mathematics, vol.152, Cambridge University Press, 2003.

[Sim70]。 Charles C. Sims, *Computational methods in study of permutation groups*, Com-putatioanl problems in Abstract Algebra, Pergamon Press, Oxford, 170, pp.169-183.

[Sim03] ______*計算群論*、計算機代数ハンドブック（Jo-hannes Grabmeier, Erich Kaltofen, and Volker Weispfenning, eds.), Springer, http://www.math.rutgers.edu/˜sims/publications/, 2003.

[TC36] J.A. Todd and H. S. M. Coxeter, *A practical method for enumerating cosets of a finite abstract group*, Proceedings of Edinburgh Mathematical Society, vol.5, 1936, pp.26-34.